Transformational design of distributed systems

S. Fickas & D. Tiktin
Dept. of Computer Science
University of Oregon, Eugene OR 97403
{fickas,tiktin}@cs.uoregon.edu

M. S. Feather & D. Cohen
USC/ISI, 4676 Admiralty Way
Marina del Rey CA 90292
{feather,donc}@isi.edu

Our position is founded upon three premises. The first is that *transformational techniques can provide useful support to the design of distributed systems.* In this approach, design commences with an idealized specification (expressed in a non-distributed fashion), and terminates with a distributed implementation that exhibits the same, or acceptably close, functionality as the specification. The design steps are accomplished by transformations; these encode commonly recurring means of translating uses of specification concepts into equivalent (or, more typically, acceptably close to equivalent) implementations. These transformations must deal with the concerns that dominate distributed systems, namely global vs. local information and control, overall system reliability in the face of failure of individual components and/or communication between them, security/privacy needs, and system-wide efficiency.

We recognize that we can rarely expect to realize a perfect (totally reliable, fully secure, maximally efficient) implementation. The concerns of distributed systems combine to make this all the more unattainable. This leads to our second premise, the need to employ *imperfect* transformations that potentially compromise or approximate the specification in the course of design; in this respect we deviate from the mainstream approach to program transformation.

Furthermore, distributed systems are often fielded in a highly variable and unpredictable environment. This, combined with the use of imperfect transformations, motivates our third premise, that *distributed systems must often be self-monitoring.* By this we mean that the system monitor its behavior and performance during its interactions with its environment. Such monitoring may reveal whether, in what form, under what conditions, and how frequently, the implementation is not completely faithful to the idealized specification. This information can be provided to the designer, who may choose to redesign the system in light of the additional understanding gained from this information, to the users, who may need to be alerted when they are using the system in a manner for which its present design is not well suited, and/or to the system itself, so that it can adapt its own behavior accordingly. Such self-monitoring is best introduced during the design process itself, rather than being tacked on after design in an ad-hoc manner. We believe this not only because we think that monitoring, like any other function, is amenable to transformational design, but also because of the especially close intertwining between the compromises and approximations that are made during the design, and the need to monitor the need for and/or consequences of such design steps.

Our studies have focussed on the early stages of system design, during which the overall architecture and interfaces are established. In particular, we consider alternative assignments of *'responsibility'* for invariants to various components of the system. Responsibility for an invariant implies acting so as to ensure its satisfaction. In a distributed setting, this frequently induces the need to cause or inhibit non-local actions, and to access non-local information. These needs in turn specify and motivate the development of interfaces between components. The concepts of responsibility, information and influence also serve to characterize the compromises and approximations (e.g., a central component specified to have knowledge of the current status of its neighbors might be approximated in a design that has those neighbors inform the central component of changes to their status; thus the central component will not necessarily have current knowledge, but instead is kept as up-to-date as possible, modulo communication delays).

We look to traditional concurrency work to provide implementations, analyses and verifications of these non-local influence and information accesses. By encoding these implementations and the responsibility manipulations as transformations, we expect that a designer will be able to readily explore the space of possible designs, and rapidly carry through any chosen design path from specification to implementation.